本次實作要將第二次實作的程式碼，用類別與繼承重新組建。架構如下:

主要有兩個類別，Crank-Nicolson Solver和Uniform grid，

Crank-Nicolson Solver

Uniform grid

Crank-Nicolson Solver繼承Uniform grid，粗略表格如下:

|  |  |  |
| --- | --- | --- |
|  | Crank-Nicolson Solver | Uniform grid |
| 負責 | solver():解出答案  output():將資料輸出成csv | grid():做出space的網格 |
| 輸入參數 | Xm,Xn,N,D,dt,f,u1,u2,(u) | Xm,Xn,N |
| 備註 | Xm=Xmax,Xn=Xmin,N=n,D=diffusion constant,dt=Δt,f=initial conditions,  u1 u2=Dirichlet Boundary Conditions,u=um-1  u格式為N\*1矩陣，可以不要輸入，預設為[]，之後有詳細說明 | |

第一要創建Uniform grid這個class，首先用\_\_init\_\_定義他的三個變數:Xm,Xn,N，此外我新增一個運算後的參數self.h=(Xm-Xn)/(N-1)，再來，定義grid()這個方法，用for將space的網格做成一個list。最後，定義solver()這個方法，直接pass，並於他的上方用@abstractmethod標註，同時，設定Uniform grid的metaclass為ABCMeta，當達成這兩個條件，就能確保之後在執行class時，會先跑過solver()這個方法，如果不能滿足solver()的話，就會跑出error。

第二要定義C\_N\_Solver，他的參數有Xm,Xn,N,D,dt,t,f,u1,u2,u=[]，

u為um-1的N\*1 array，如果沒有輸入的話則自動設為空的list，我一樣用\_\_init\_\_定義的他參數，值得注意的是，他的Xm,Xn,N是由UniformGrid繼承而來，所以要用UniformGrid.\_\_init\_\_(self,Xm,Xn,N)。

接著，我定義了solver這個函式以進行求值的運算，先用if判定u是否為空的array，如果為空的，則用f(initial conditions)和grid做出u這個矩陣，值得注意的是，這邊的grid前面要用super()這個方法，因為super()是用來表達被繼承類別的函式。接著就如同我結報二中所述，創建a為左邊的矩陣，b為右邊的矩陣，用np.linalg.solve(a,b)去解出答案，我設定答案的矩陣為e，不過，為了滿足boundary conditions ，我指定e在[0,0]位置的值為u1，e在[N-1,0]位置的值為u2，最後指定self.u=e，以利於之後進行疊帶運算。

再來我設定output，我在裡面定義time函式，參數為t,dt,N，目的要做出t+dt的N個元素的list。接著，我用super().grid()將空間的資料取出，用list(u.ravel())將solution的list取出。這邊說明取出solution的list的方法，先用ravel()將array裡面的元素降階一次，再用list函式將轉換後的array轉換成N\*1的list。

得到time,space,solution的三個list後，我先把三個小list寫進一個大list，再轉成array的格式並轉置此矩陣，再將此矩陣轉成dataframe的格式，用to\_csv去輸出結果，to\_csv的 mode設定為’a’(add)的原因在於:這樣有利於之後疊帶運算輸出結果，也設定不需要標題:header=None。

測試題目為Xmax=1,Xmin=0, Δt=0.001,f(t=0)=sin(πx),f(x=0)=f(x=1)=0,n=51,，要算出t=0.009的值。所以我用solver運算九次並將結果append進一個空的list，最後，觀察這個list的最後一個元素，即為本次的結果。

至於output的方面，也是用for去達成，只是在for中需要指定u=solver()做出的結果，而且t在每次運算中也要換成t+dt，就能順利的用output()輸出了。